计算机图形学 HW1

21307289 刘森元

1. 实验环境

Macbook Pro 14 inchs, 2021 (Apple M1 Pro)

macOS Ventura 13.5.2

qt: stable 6.6.0 (bottled), HEAD

2. 环境搭建

安装 Homebrew

```
> /bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
> brew --version
Homebrew 4.1.17
```

安装 qt

```
> brew install qt
> brew info qt
==> qt: stable 6.6.0 (bottled), HEAD
Cross-platform application and UI framework
https://www.qt.io/
/opt/homebrew/Cellar/qt/6.6.0 (14,393 files, 623.2MB) *
  Poured from bottle using the formulae.brew.sh API on 2023-11-09 at 01:37:01
From: https://github.com/Homebrew/homebrew-core/blob/HEAD/Formula/q/qt.rb
License: BSD-3-Clause and GFDL-1.3-no-invariants-only and GPL-2.0-only and (GPL-3.0-only with
Qt-GPL-exception-1.0) and LGPL-3.0-only
==> Dependencies
Build: cmake ✔, ninja ✘, node ✔, pkg-config ✔, python@3.11 ✔, six ✔, vulkan-headers ✘,
vulkan-loader ✘, molten-vk ✘
Required: assimp ✔, brotli ✔, dbus ✔, double-conversion ✔, freetype ✔, glib ✔, harfbuzz ✔,
hunspell ✔, icu4c ✔, jasper ✔, jpeg-turbo ✔, libb2 ✔, libmng ✔, libpng ✔, libtiff ✔, md4c ✔,
openssl@3 ✔, pcre2 ✔, sqlite ✔, webp ✔, zstd ✔
==> Requirements
Build: Xcode (on macOS) ✘
==> Options
--HEAD
        Install HEAD version
==> Caveats
You can add Homebrew's Qt to QtCreator's "Qt Versions" in:
  Preferences > Qt Versions > Link with Qt...
```

```
pressing "Choose..." and selecting as the Qt installation path:
  /opt/homebrew
==> Analytics
install: 31,857 (30 days), 88,796 (90 days), 198,558 (365 days)
install-on-request: 21,259 (30 days), 56,692 (90 days), 142,869 (365 days)
build-error: 82 (30 days)
```

使用 qt

使用如下命令进行项目构建

```
> qmake -project
> qmake
> make
```

3. 作业内容

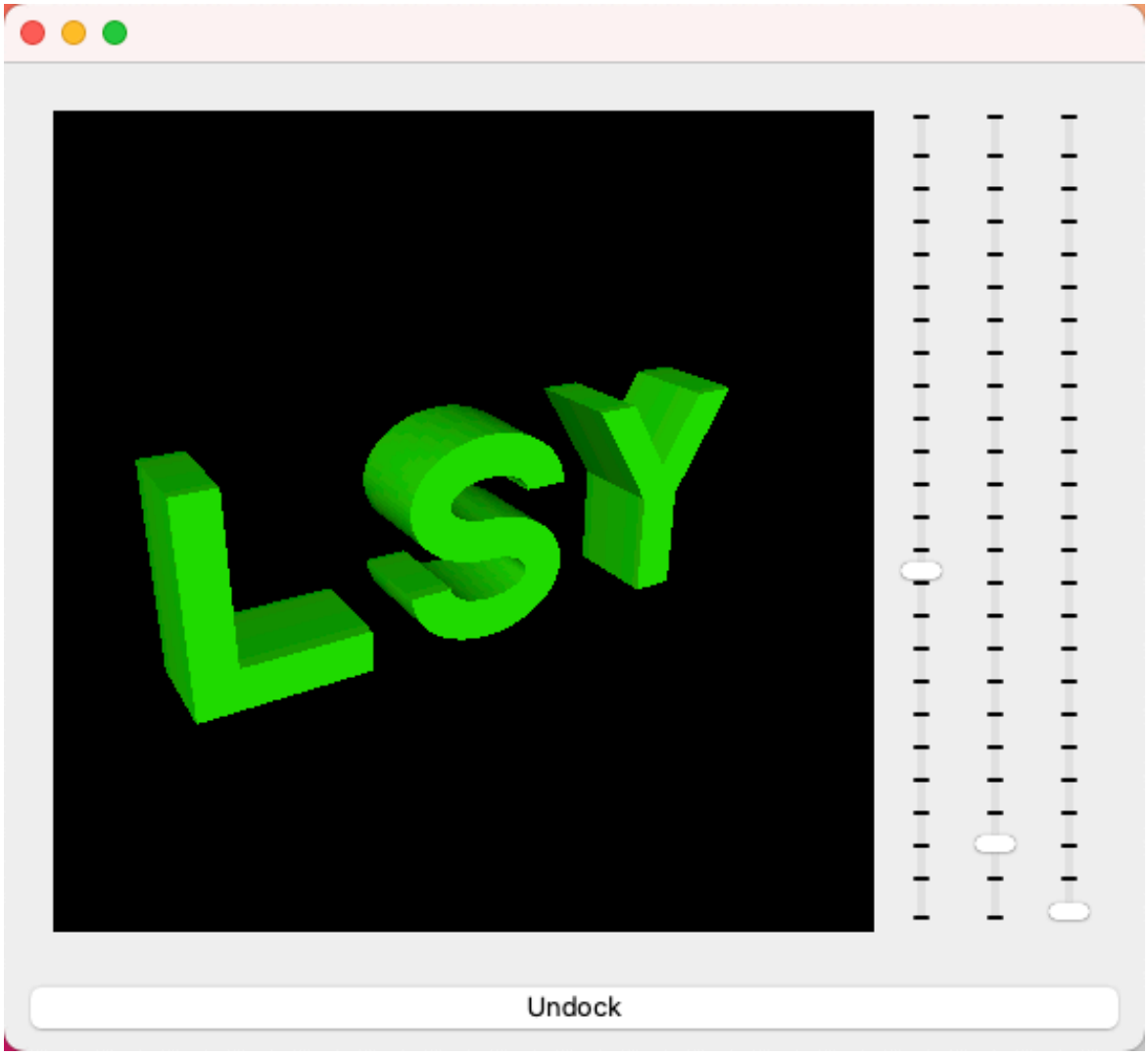
由于绘制平面首字母与立体首字母大同小异，此处不再赘述

当前项目代码基于 https://github.com/qt/qtbase/tree/dev/examples/opengl/hellogl2 构建

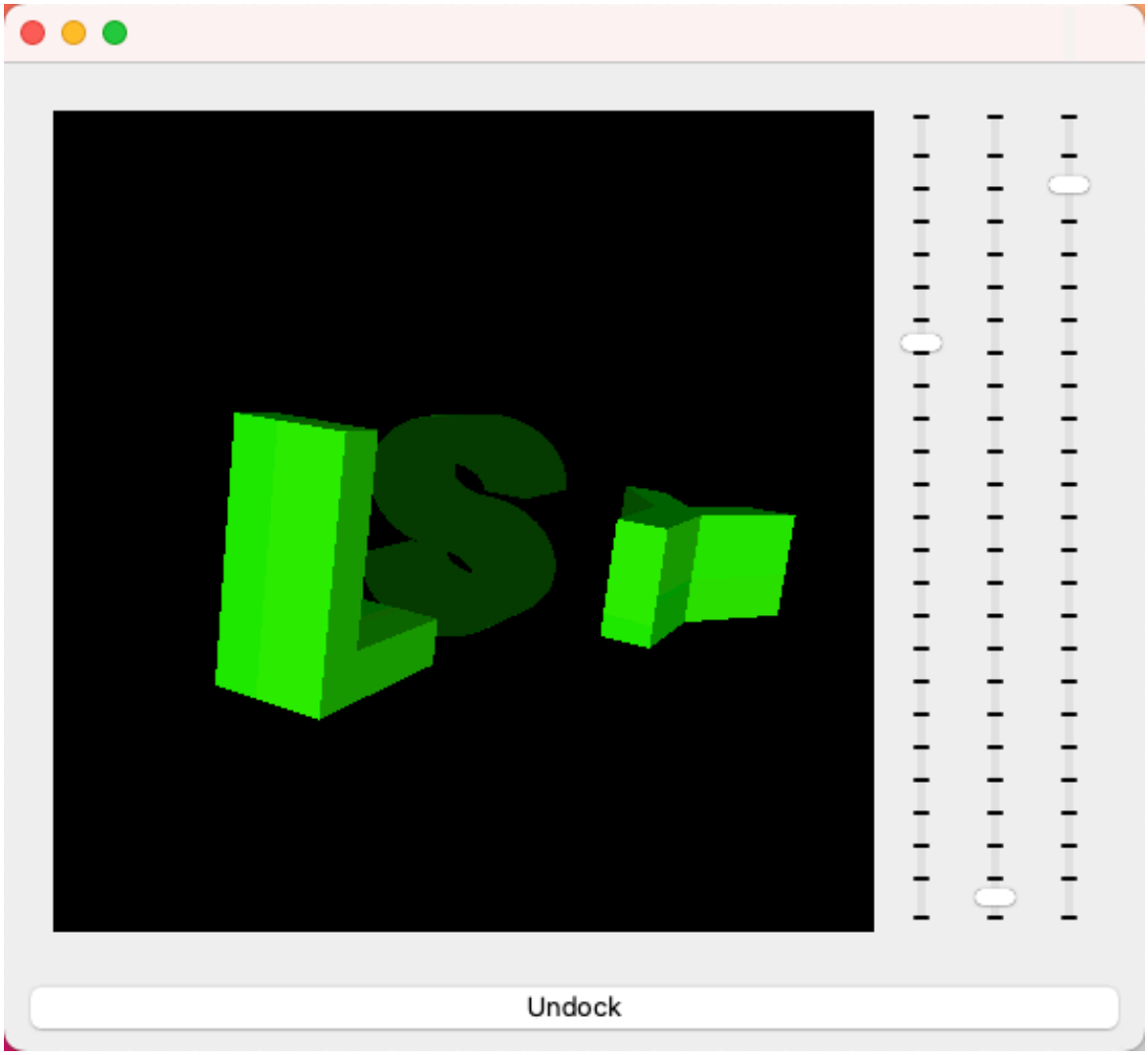
1] 结果展示

动态效果展示详见 video.mp4

1）字母位于同一平面



2）字母位于不同平面



2] 功能实现

1）绘制图形

构建类 `Logo`，其中按照 `(x, y, z)` 的顺序存储顶点进行绘制

class logo

```
class Logo {
public:
    Logo(); // 构造函数
    const GLfloat *constData() const { return m_data.constData(); } // 返回数据指针
    int count() const { return m_count; } // 返回数据数量
    int vertexCount() const { return m_count / 6; } // 返回顶点数量

private:
    void quad(GLfloat x1, GLfloat y1, GLfloat x2, GLfloat y2, GLfloat x3, GLfloat y3, GLfloat
x4, GLfloat y4, const int &axis); // 绘制顶面和底面
    void extrude(GLfloat x1, GLfloat y1, GLfloat x2, GLfloat y2, const int &axis); // 绘制侧面
    void add(const QVector3D &v, const QVector3D &n, const int &axis);

    QList<GLfloat> m_data; // 存储顶点数据的列表
```

```
    int m_count = 0; // 当前顶点数量
};
```

void add()

```
void Logo::add(const QVector3D &v, const QVector3D &n, const int &axis) {
    GLfloat *p = m_data.data() + m_count;

    // 通过 axis 来决定定点输入顺序，以确定绘制平面
    if (axis == 0)
        *p++ = v.z();
    *p++ = v.x();
    if (axis == 1)
        *p++ = v.z();
    *p++ = v.y();
    if (axis == 2)
        *p++ = v.z();

    if (axis == 0)
        *p++ = v.z();
    *p++ = n.x();
    if (axis == 1)
        *p++ = v.z();
    *p++ = n.y();
    if (axis == 2)
        *p++ = v.z();

    m_count += 6;
}
```

最后通过 `glDrawArrays(GL_TRIANGLES, 0, m_logo.vertexCount())` 遍历顶点进行绘制

具体字母绘制详见 Logo::Logo() 实现

2）坐标轴旋转

通过绘制的线性变换进行旋转

glwidget.cpp

```
m_world.setToIdentity();
m_world.rotate(m_xRot / 16.0f, 1, 0, 0);
m_world.rotate(m_yRot / 16.0f, 0, 1, 0);
m_world.rotate(m_zRot / 16.0f, 0, 0, 1);
```

3] 讨论内容

讨论 1:

1. GL_TRIANGLES:
   - 绘制开销：每个三角形需要3个顶点，因此需要的glVertex调用次数是顶点数的三倍。
   - 示例代码：

```
glDrawArrays(GL_TRIANGLES, 0, vertexCount);
```

2. GL_TRIANGLE_STRIP:
   - 绘制开销：每个三角形需要3个顶点，但是后续的顶点可以通过重用前面的顶点来减少glVertex调用次数。对于n个顶点，需要的glVertex调用次数是n+2。
   - 示例代码：

```
glDrawArrays(GL_TRIANGLE_STRIP, 0, vertexCount);
```

3. GL_QUAD_STRIP:
   - 绘制开销：每个四边形需要4个顶点，但是后续的顶点可以通过重用前面的顶点来减少glVertex调用次数。对于n个顶点，需要的glVertex调用次数是2n。
   - 示例代码：

```
glDrawArrays(GL_QUAD_STRIP, 0, vertexCount);
```

讨论 2:

1. 从(0,0,d)看向原点(0,0,0):
   - Orthogonal投影方式：在正交投影中，远近物体的大小和比例是相同的，不受距离的影响。因此，无论观察者距离原点多远，图像中的物体都将保持相同的大小和比例。
   - Perspective投影方式：在透视投影中，远处的物体看起来比近处的物体小。因此，当从(0,0,d)观察原点时，离观察者更远的物体将显得更小，而离观察者更近的物体将显得更大。
2. 从(0,0.5*d,d)看向原点(0,0,0):
   - Orthogonal投影方式：在正交投影中，无论观察者的位置如何，远近物体的大小和比例都保持不变。因此，无论观察者位于何处，图像中的物体都将具有相同的大小和比例。
   - Perspective投影方式：在透视投影中，观察者的位置会影响远近物体的大小和比例。当从(0,0.5*d,d)观察原点时，离观察者更远的物体将显得更小，而离观察者更近的物体将显得更大。

综上所述，Orthogonal投影方式下，远近物体的大小和比例保持不变；而Perspective投影方式下，远处的物体看起来比近处的物体小。具体的效果取决于观察者的位置和投影方式的设置。